Statistical Techniques in Robotics (16-831, S21) Lecture #11 (Wednesday, March 10)

AdaBoost & MAB

Lecturer: Kris Kitani *Scribes: Jinkun Cao, Yuda Song*

1 Review

In the last lectures, on the problem of supervised learning, we studied some online technique SVM, i.e. Support Vector Machine. Online techniques can be very useful for supervised learning. Different from the usual supervised learning, online learning interweaves the training and testing steps.

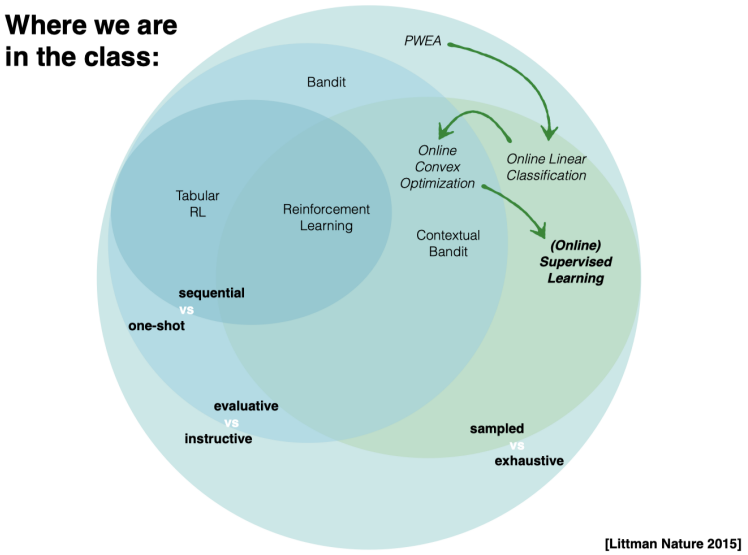

Figure 1: Where the online supervised learning is categroied.

1.1 Support Vector Machine (SVM)

1.1.1 Hyperplanes

To have a good classifier, we basically want to find a proper hyperplane to divide data points into different "regions". A hyperplane is a subspace whose dimension is one less than that of its ambient space. If a space is 3-dimensional then its hyperplanes are the 2-dimensional planes, while if the space is 2-dimensional, its hyperplanes are the 1-dimensional lines[1]. A common example for the most seen 1D and 2D hyperplanes are shown in Figure 2 and Figure 3.

[1] https://en.wikipedia.org/wiki/Hyperplane

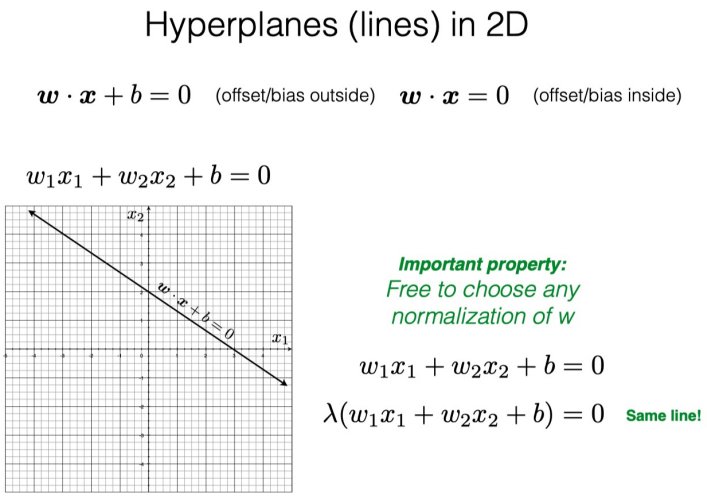

Figure 2: A hyperplane in 2D space is just an 1D line.

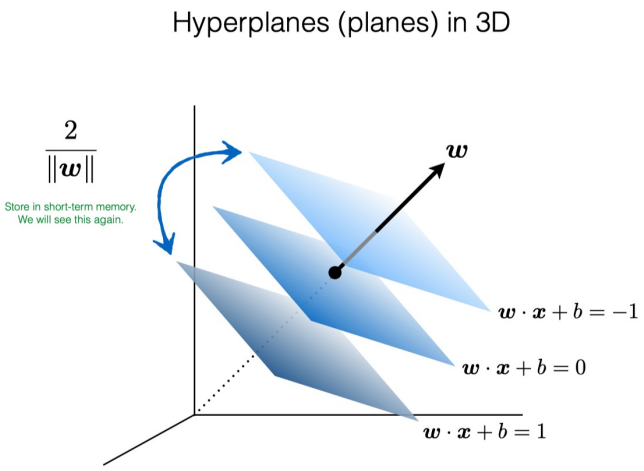

Figure 3: A hyperplane in 3D space is just an 2D plane.

1.1.2 Support Vector Machine

A Support Vector Machine (SVM) [7] is basically a max-margin classifier. It aims to find a hyperplane in the data space to divide data points into categories they belong to. And the optimization objective is usually to maximize the margin between data points and the hyperplane boundary. Consider we describe the hyperplane with

$$\mathbf{w} \cdot \mathbf{x} + b = 0, \tag{1}$$

and the points that most close to the plane on two sides are located on the hyperplane $\mathbf{w} \cdot \mathbf{x} + b = 1$

and $\mathbf{w} \cdot \mathbf{x} + b = -1$. Then, for the two-class classification setting, we can formulate the objective as a maximization problem, for $i = 1, ..., N$,

$$max_{\mathbf{w}} \frac{2}{||\mathbf{w}||}, \quad s.t. \quad \mathbf{w} \cdot \mathbf{x}_i + b \quad \begin{cases} \geq +1 if y_i = +1 & (2) \\ \leq -1 if y_i = -1. & (3) \end{cases}$$

So it is equivalent to find

$$min_{\mathbf{w}} ||\mathbf{w}||^2, \quad s.t. \quad y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 (i = 1, ..., N), \tag{4}$$

which is a convex quadratic programming problem, so there exists a unique solution.

1.1.3 'Soft' Margin SVM

In practice, there might be some situations where we can not find a hyperplane to divide points of different categories into different sides perfect or there exists only extremely narrow margin if we force such a "perfect" classification which might lead the solution less robust because of overfitting. So intuitively, we should allow for some misclassification if we can get more robust classification. Under such a philosophy, we actually make a trade-off between the **MARGIN** and the **MISTAKES**. To realize this, we add a slack variable ξ, making the constrain,

$$y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1 - \xi_i. \tag{5}$$

By allowing the slackness, we convert the original SVM into the 'soft' margin SVM. The objective is converted to

$$min_{\mathbf{w},\xi} ||\mathbf{w}||^2 + C \sum_i \xi_i, \tag{6}$$

where C is a regularization parameter that a small C tends to ignore the constrain to find a large margin while a big C tends to emphasize the effect of constrain. By adding the slackness, the optimization objective still forms a quadratic programming problem, thus an unique solution exists. A illustration of 'Soft' Margin SVM is shown in Figure 4.

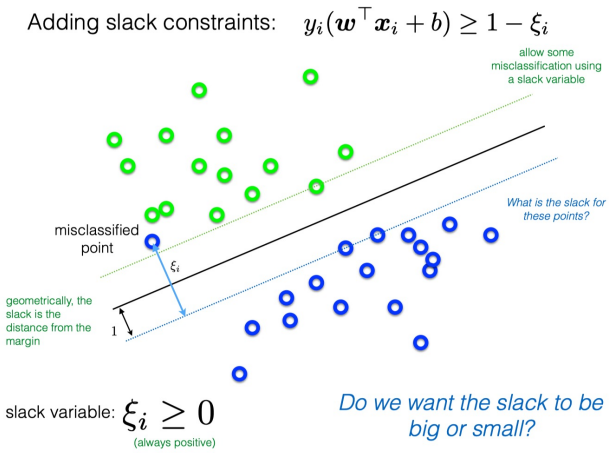

Figure 4: When adding slackness into SVM, we allow some misclassification to make the solution more robust.

Finally, by combining the regularization and loss function, we have an adaptive optimization object

$$min_{\mathbf{w}}(\frac{\lambda}{2}||\mathbf{w}||^2 + \frac{1}{M}\sum_{m=1}^{M} max\{0, 1 - y_m \mathbf{w}^T \mathbf{x}_m\}), \tag{7}$$

which is convex but not differentiable. Hence, to solve the problem, we use Sub-Gradient Descent instead of the Gradient Descent we have been familiar with. A subgradient[2] for a convex function f at x is any $g \in \mathbb{R}^n$ such as

$$f(y) \geq f(x) + g^T(y - x). \tag{8}$$

For the hinge loss, we have its subgradient expression as

$$z_m = \begin{cases} \mathbf{0}, & y_m \mathbf{w}^T \mathbf{x}_m \geq 1 \quad (9) \\ -y_m \mathbf{x}_m, & otherwise. \quad (10) \end{cases}$$

So come to here, we could write the algorithm of soft margin SVM as in Algorithm 1.

Algorithm 1 Soft SVM

1: $\theta^{(1)} \longleftarrow 0 \in \mathbb{R}^N$
2: **for** t=1,...,T **do**
3: $y_d, x_d \sim D$
4: $\theta^{(t)} = \theta^{(t-1)} + y_d x_d \cdot \mathbf{1}[y_d(\mathbf{w}^{(t)} \cdot \mathbf{x}_d) < 1]$
5: $\mathbf{w}^{(t+1)} \longleftarrow \frac{1}{\lambda(t+1)}\theta^{(t)}$
6: **end for**
7: $\bar{\mathbf{w}} = \frac{1}{T}\sum_t \mathbf{w}^t$

[2]https://www.stat.cmu.edu/ ryantibs/convexopt-F15/lectures/06-subgradients.pdf

2 PAC Learning

PAC Learning [12], short for "Probably Approximately Correct Learning", is a theoretical framework for answering the question: **What is the optimal dataset size to obtain good generalization?**

To explain the learning framework, we need some terminology first. When a dataset D of size N is drawn from a distribution $P(x, y)$ and the class labels are determined by an unknown deterministic distribution $y = f^{\star}(x)$. Then for a single classifier, we could denote a function f drawn from space of function $\mathcal{F}$ and obtained by training on the training data, so we have

$$f(x; D) \sim \mathcal{F}|D. \tag{11}$$

For a learned function f, it has good generalization if its expected error rate is below a pretrained value

$$E_{p(x,y)}[\mathbf{1}[f(x; D) \neq y]] < \epsilon. \tag{12}$$

Under these backgrounds, a PAC learning algorithm requires that the inequality 12 holds with probility $1-\sigma$, where ϵ is defined to define a "approximately correct". Broadly, there are two types of PAC learning algorithms: Strong PAC-learning and Weak PAC-learning.

Strong PAC-learning Algorithm. Given ϵ, σ and $D \sim P(x, y)$, learner outputs with probability $1 - \sigma$ a hypothesis (function) with error at most ϵ.

Weak PAC-learning Algorithm. Given ϵ, σ , $D \sim P(x, y)$ and $\gamma > 0$, learner outputs with probability $1 - \sigma$ a hypothesis (function) with error at most $\epsilon \geq 1/2 - \gamma$.

For both strong and weak PAC-learning algorithms, the run time must to polynomial in $1/\sigma$ and $1/\epsilon$ and other relevant parameters such as the size of the examples or complexity of the target concept.

Then, the key point is ... **Any weak learning algorithm can be boosted into a strong learning algorithm** [9].

3 AdaBoost

To have a stronger PAC learner by boosting from weak learners, the basic strategy is to **call a weak PAC learner multi times (generates a sequence of hypothesis), but at each time, present it with a different distribution D. Then combine all hypothesis into a single hypothesis (strong PAC learner).**

Following the philosophy, AdaBoost [3], short for "Adaptive Boosting", it is a machine learning to be used in conjunction with other types of learning algorithms to improve (boost) performance. The output of the other learning algorithms is combined into a weighted sum that represents the

final output of the boosted version of classifier [3]. The algorithm of AdaBoost is presented in Algorithm 2. Where line 3-4 are for prediction from the weak learner and line 5-7 are for update parameters.

Algorithm 2 AdaBoost

Require: $D = \{x_n, y_n\}_{n=1}^{N}$, $\{w_n^{(0)}\}_{n=1}^{N}$, T

1: **for** t=1,...,T **do**
2: $\quad \mathbf{p}^{(t)} = \mathbf{w}^{(t-1)} / \sum_n w_n^{(t-1)}$
3: $\quad h^{(t)} = \text{WEAKLEARNER } (D, \mathbf{p}^{(t)})$
4: $\quad \epsilon^{(t)} = \sum_n p_n^t |h^{(t)}(x_n) - y_n|$
5: $\quad \beta^{(t)} = \epsilon^{(t)} / 1 - \epsilon^{(t)}$
6: $\quad w_n^{(t)} = w_n^{(t-1)} \beta^{1-|h^{(t)}(x_n^{(t)}) - y_n^{(t)}|} \forall n$
7: **end for**
8: $h_F(x) = \mathbf{1}[\sum_{t=1}^{T} (log \frac{1}{\beta^{(t)}}) h^{(t)}(x) \geq \frac{1}{2} \sum_{t=1}^{T} (log \frac{1}{\beta^{(t)}})]$

Within the algorithm, we should pay attention to some important components. $\mathbf{p}^{(t)}$ is the probability distribution over the data items. It is used to re-weight the training dataset then more weight is assigned to "important/hard" data items. $h^{(t)}$ is the hypothesis or classifier returned by a weak learning algorithm. Then, for the update part, $\epsilon^{(t)}$ is the average error/loss of the obtained hypothesis in the prediction stage. $\beta^{(t)}$ is the penalty constant to judge a classifier. For example, when the classifier is good, β should approach 0 or it approaches infinity if the classifier is bad. Finally, in line 6, the weight is updated in a manner of exponential update. Line 8 in the algorithm gives the final hypothesis after boosting where it contains an inequality of which the LHS is the weighted combination of hypothesis and the RHS is the weight total.

With this AdaBoost algorithm, the error bound is essentially

$$\epsilon \leq w^T \Pi_{t=1}^{T} \sqrt{\epsilon_t (1 - \epsilon_t)}. \tag{13}$$

4 Multi-Armed Bandit

We then come to the area of solving Multi-Armed Bandit problem where a fixed limited set of resources must be allocated between competing (alternative) choices in a way that maximizes their expected gain, when each choice's properties are only partially known at the time of allocation, and may become better understood as time passes or by allocating resources to the choice [4]. The concept of Multi-Armed Bandit problem is essentially explained in the Veen Diagram of Figure 5.

4.1 Introduction

Basically, for a typical Multi-Armed Bandit, we have multiple bandit machines, each of which has an unknown reward distribution. We are allowed to pull one arm among them to get a reward each

[3] https://en.wikipedia.org/wiki/AdaBoost
[4] https://en.wikipedia.org/wiki/Multi-armed_bandit

time. We need to carefully select which arm to pull next to maximize the total reward over a time interval or T decisions. MAB problem has

- one-shot feedback, as one action leads to one reward where selecting an action does not change the state of the next time step;
- exhaustive feedback, as the 'state' is static and finite because there is no state and actions are finite. A player could pull all arms over the duration of the game;
- evaluative feedback, as we receive a sampled reward at each time step.

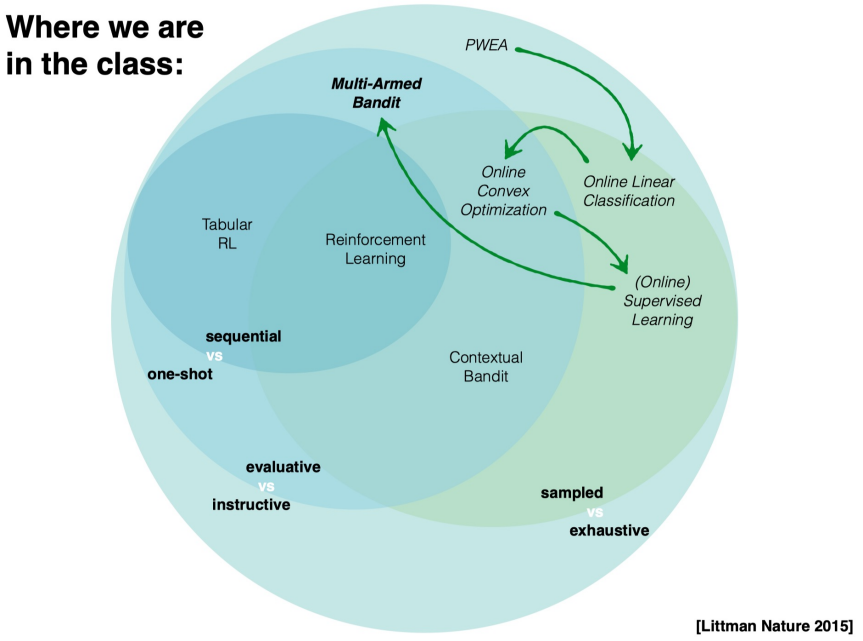

Figure 5: Where the Multi-Armed Bandit problem is in the Concept Venn Diagram.

In previous tasks, we usually study the loss from an action $l_t \in [0, 1]$, now we can rewrite it as a reward $g_t = 1 - l_t \in [0, 1]$. Then the regret now becomes

$$R^{(T)}(h) = \underbrace{\sum_{t=1}^{T} g(h(x^{(t)}), y^{(t)})}_{\text{reward of a hypothesis (expert)}} - \underbrace{\sum_{t=1}^{T} g(\hat{y}^{(t)}, y^{(t)})}_{\text{reward of the online algorithm}} . \tag{14}$$

4.2 Exploration-Exploitation

Different from PWEA situation, where we have full loss observation, we only have partial loss observability in bandit problem. It says that learner observes loss only for their action. So, we need to explore what the feedback might be if we take different actions, which leads to an Exploration-Exploitation solution naturally. Because we need to **explore arms that might do well in**

the future and **exploit arms that did well in the past.** We note here that this solution is proposed under the **Stochastic Bandits** environments [8] where (1) each arm has a static reward distribution and (2) each pull gives a sample from a distribution. The process of Explore-Exploit under careful balance of the tradeoff of exploration and exploitation is shown in Algorithm 3.

Algorithm 3 Explore-Exploit

Require: M
for $k = 1, \cdots, K$ **do**
  for $m = 1, \cdots, M$ **do**
    $a = k$
    Receive(r)
    $\widehat{\mu}_k = \widehat{\mu}_k + \frac{r}{M}$
  end for
end for
for $t = KM, \cdots, T$ **do**
  $a^{(t)} = \arg\max_k \widehat{\mu}'_k$
end for

And the regret bound for Explore-Exploit is

$$R_{\text{explore-exploit}} = \mathcal{O}(K^{1/3}T^{2/3}). \tag{15}$$

References

[1] S. Bubeck and N. Cesa-Bianchi. Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *arXiv preprint arXiv:1204.5721*, 2012.

[2] V. Cherkassky and Y. Ma. Practical selection of svm parameters and noise estimation for svm regression. *Neural networks*, 17(1):113–126, 2004.

[3] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.

[4] T. Joachims. Making large-scale svm learning practical. Technical report, Technical report, 1998.

[5] S. Kalyanakrishnan, A. Tewari, P. Auer, and P. Stone. Pac subset selection in stochastic multi-armed bandits. In *ICML*, volume 12, pages 655–662, 2012.

[6] O.-A. Maillard, R. Munos, and G. Stoltz. A finite-time analysis of multi-armed bandits problems with kullback-leibler divergences. In *Proceedings of the 24th annual Conference On Learning Theory*, pages 497–514. JMLR Workshop and Conference Proceedings, 2011.

[7] W. S. Noble. What is a support vector machine? *Nature biotechnology*, 24(12):1565–1567, 2006.

[8] H. Robbins. Some aspects of the sequential design of experiments. *Bulletin of the American Mathematical Society*, 58(5):527–535, 1952.

[9] R. E. Schapire. The strength of weak learnability. *Machine learning*, 5(2):197–227, 1990.

[10] S. L. Scott. A modern bayesian look at the multi-armed bandit. *Applied Stochastic Models in Business and Industry*, 26(6):639–658, 2010.

[11] S. Shalev-Shwartz, Y. Singer, N. Srebro, and A. Cotter. Pegasos: Primal estimated sub-gradient solver for svm. *Mathematical programming*, 127(1):3–30, 2011.

[12] L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

[13] J. Vermorel and M. Mohri. Multi-armed bandit algorithms and empirical evaluation. In *European conference on machine learning*, pages 437–448. Springer, 2005.

5 Appendix

We provide some materials here which might be interesting to read and highly related to this lecture.

5.1 More for SVM

Despite the elegant form of SVM algorithms, it is hard to solve it in many real scenes, when, for example, the data scattering is messy to obtain good enough hyperplane with convincing margin or the data volume is too large to design efficient solver and so on. To tackle these real-world problems, many researchers have provided great progress in the application of SVM.

An early paper [4] focuses on the quadratic optimization problem with bound constraints and one linear equality that training a SVM leads to. From this reason, large learning tasks with many training examples on the shelf optimization techniques for general quadratic programs quickly become intractable in their memory and time requirements. Authors of this paper designed a system to solve large-scale SVM training in an efficient manner.

Another work [2] continues to investigate practical selection of hyper-parameters for support vector machines (SVM) regression (that is, ϵ-insensitive zone and regularization parameter C we introduced in this lecture). The proposed methodology advocates analytic parameter selection directly from the training data, rather than re-sampling approaches commonly used in SVM applications. They also prove the good generalization ability of the proposed scheme.

A more recent work [11] makes a breakthrough on solving SVM more quickly. They describe and analyze a simple and effective stochastic sub-gradient descent algorithm for solving the optimization problem cast by Support Vector Machines (SVM). We prove that the number of iterations required to obtain a solution of accuracy ϵ is $O(1/\epsilon)$, where each iteration operates on a single training example. In contrast, previous analyses of stochastic gradient descent methods for SVMs require $\Omega(1/\epsilon^2)$ iterations.

5.2 More for Multi-Armed Bandits

On a broad range of Multi-Armed Bandits, there are actually diverse works analysing solution complexity, performance bound and proposing new solutions.

5.2.1 More Analysis

A very famous paper [1] focuses on two extreme cases in which the analysis of regret of MAB problem is particularly simple and elegant: i.i.d. payoffs and adversarial payoffs. Besides the basic setting of finitely many actions, they also analyze some of the most important variants and extensions, such as the contextual bandit model. More recently, a work [13] studies the performance of existing solutions for Multi-Armed Bandits in an empirical perspective.

From the perspective of KL divergence, a previous work [6] proposes a finite-time analysis of Multi-armed Bandits Problems, whose conclusion agrees with previous analysis from different perspectives.

5.2.2 More applications

Besides, for some special cases of MAB problem, there are also some interesting works. For example, for the generalization of single-armed bandit problem to the classic subset selection problem, a previous work [5] addresses and analyses it pretty well. It gives a novel expected sample complexity bound for subset (and single-arm) selection. They have also given a worst case sample complexity lower bound for subset selection.

Some literature also tries to combine other tools with existing knowledge in this area. For example, a recent paper [10] tries to leverage Bayesian computation in this area. As the advances in Bayesian computation have made randomized probability matching easy to apply to virtually any payoff distribution. This flexibility frees the experimenter to work with payoff distributions that correspond to certain classical experimental designs that have the potential to outperform methods that are 'optimal' in simpler contexts. in this paper, the authors summarize the relationships between randomized probability matching and several related heuristics. And its context is pretty modern that the focus is mainly on the application in reinforcement learning.